A Machine Learning Solution to a Marketing Problem: the Recommender System

Master's thesis

Author:
Félix Revert

Advisors:
Prof. Dr. Ostap Okhrin
Prof. Dr. Weining Wang

Humboldt Universität

CASE - Center of Applied Statistics and Economics

September 26, 2014

Declaration of Authorship

I hereby confirm that I have authored this master thesis independently and without use of others than the indicated resources. All passages, which are literally or in general matter taken out of publications or other resources, are marked as such.

Félix Revert

Berlin, September 25, 2014

Abstract

This essay presents a marketing problematic along with a way of solving it with the help of statistical tools. Precisely, a Recommender System is built to address a marketing problem for a supermarket in Germany. Both the theoretical and the practical aspects of the use of Machine Learning methods are underlined. Our focus is oriented in two parts. First, we define precisely a problem, from the conceptual meaning to its natural mathematical translation. Second, we provide an analysis that is anchored to business constraints, on top of being an academic paper. Highlight is made on unifying notations for this specific problem, and on comparing different methods before and after their implementation. This work is designed to fulfill the Master's thesis requirements with advanced statistical approach. Mathematical background is emphasized in parallel with hands-on implementation using the open source software `R`.

Contents

1 Introduction

The field of Machine Learning has been booming over the last decade. With both computers' increasing capacity and the rise of the amount of data stored, Machine Learning has proven an acute tool for both understanding the structure of a dataset and for predicting accurately from it. A Recommender System, as explained in the section 4, uses statistical analysis to somehow rank the products in order to recommend the ones at the top of the list, which are the most relevant for some defined purpose. Currently it is one of many applications of Machine Learning, still we are far from knowing everything about it when one knows the variety of data that is collected nowadays. Many companies indeed like Walmart or Amazon use such technology to generate personalized recommendations to their clients, but there may be a colossal number of ways to do it. As a matter of fact, these two companies use very different algorithms since both the people who are developing it are different and the datasets are different as well.
This work also provides its own unicity. It is intended to follow the exact process of a company that has to solve a problem: first, the description of the issue, seconds the methods to tackle it, third, the final solution(s) with its remarks. Therefore the plan of this work presents itself as these different chronological phases of the process.
In the first part I discuss in details the problematics of the topic. In the second part I enumerate some of the existing techniques in Machine Learning and analyse them with regard to the problem. In the last part I present the results of the algorithms that were implemented, investigate the obtained results and then conclude this work.

2 A Marketing Problem

2.1 Purpose of the Problem

2.1.1 A brief description

A German supermarket wishes to use a new concept for its customers. It consists in having machines printing vouchers working in connection with loyalty cards. This type of machine already exists, yet the concept lies in the computer system that chooses smartly which products to print coupons for a given customer. Precisely, the aim is to select products in a way that it would increase the size of the customer's basket. Therefore, the goal is to recommend products that do not interfere with the usual customer's purchases, that do not shift the customer's habits, but extend them.
Underlying questions arise from this idea. For instance, the incentive for customers to buy new products would lie in the price reduction that is offered to them. One might suppose that a discount limit in the price of a specific product exists for every person. This limit stands for the turning point to which a customer will start purchasing the product discounted. This price limit is very likely to differ from a person to another.
Another concern arises when a discount is given for a product that is unneeded. This will certainly not result in a purchase. In other words, printing the voucher is a failure because if a more relevant product discount was given, a purchase would have been induced.
But more important, printing a coupon for a product to a customer who had already the intention to buy will result in a loss for the supermarket, valuing to the amount of the discount.
As a consequence, this idea to create personalized coupons unveils some complications. If the concept seems all right in the first place, one must be very careful about the undesirable consequences it can lead to. But this is also an

incentive to use Machine Learning to address this problem since it provides a wide range of methods to tackle this type of problems.

2.1.2 The problem in detail

As I underlined previously, the idea presents different issues concerning both the selection of the products to present to a customer, and the amount of discount for each one of the products relative to the customer. Obviously these two issues are very linked.
Let us face the problem another way. Suppose our dataset contains the history of purchases of one of our customer. We know exactly to which retailer he bought which product, for which price and at which time. If we are to give one voucher to this customer, which product and which discount will we pick ? We know exactly which products he does not buy in our supermarket, and at which price. Therefore it makes sense to select products that are already in the customer's history of purchase if we are to maximize our chance the customer will use the coupon. And we would choose a discount that makes the price a bit lower to what he is used to in the other supermarket where buys the product in question.
If the question of selecting the 'good' products is simple to answer, choosing the right amount of discount that will turn the voucher to a purchase is a tougher one. Furthermore, the discount can be seen as a psychological incentive and picking the right amount is much harder to grasp with a mathematical approach than picking the products.
Our dataset is very similar to the fictive one presented above, and we face these exact questions: which products, and which discounts. As the problem of choosing the discount is too complex, the methodology I use will be focused only on the selection on the products. The discount is still important, but it extends the problem far beyond the field of statistics. In our work we suppose their is already a method for discounting the product to recommend. In practice, the amount of discount would be the same for every product so to pinpoint only the recommendation method. First let us define a bit more the entities we are working with:

- Retailer 1, which is the supermarket that wants to implement this personalized recommendation system. This retailer provides its loyalty card data.

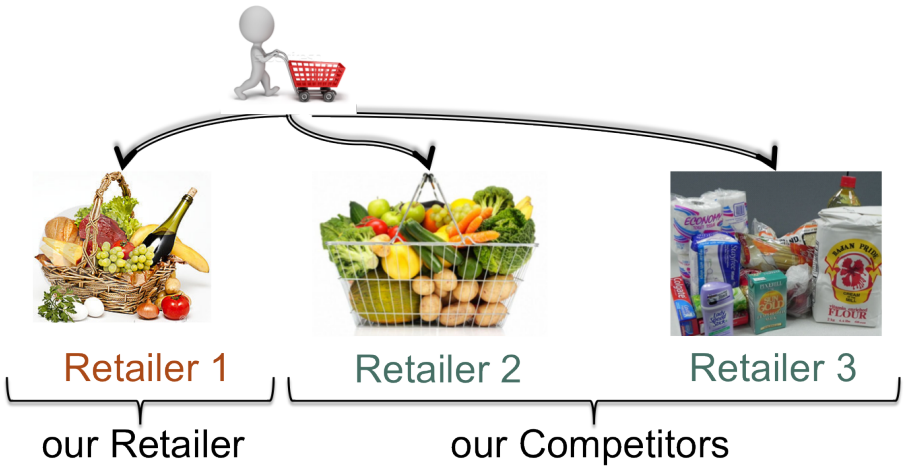


Figure 2.1: The usual behavior of a customer of Retailer 1: shopping is not restricted to Retailer 1

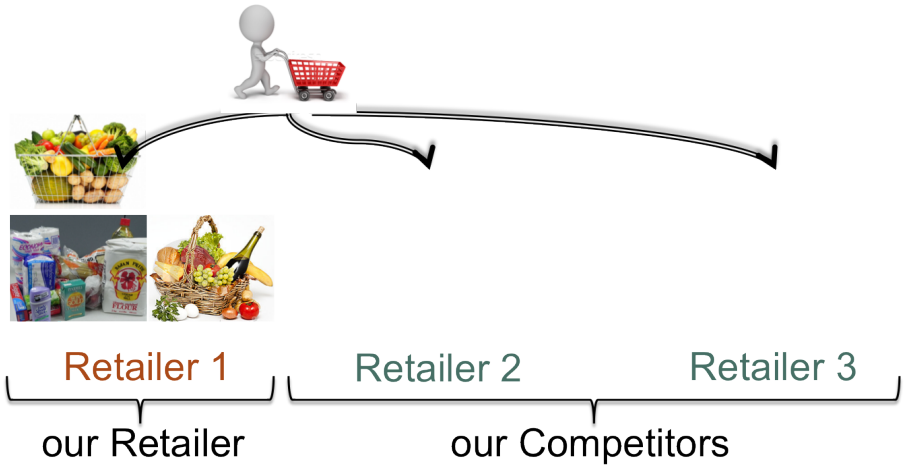


Figure 2.2: The ideal goal of the Recommender System: shifting the purchases of the customer

- All other retailers, which represent all other competitors of Retailer 1 in Germany.
- the Data provider, which provides the data of the history of purchases of a panel of households in Germany. We will discuss later the core of the data.

As said earlier, the aim of this recommender system is to make customers purchase products on top of their usual basket. Products recommended should be known by customers. This for instance is very different to other recommender systems, as the ones of Netflix or Amazon. The number of recommendations is also restricted, in practice 8 coupons will be printed per customer (and per day). From now on I will use the terminology Recommender System for the system to be built.

2.2 Business Conditions

This work has been done in a business environment. Actually, I have closely worked with a company (that wishes to remain anonymous). The hands-on part of the work lasted one full month while previous meetings prepared the ground for it.
The aim was thus to create a RS that could be used directly by the supermarket.

2.2.1 Trade-offs

In every Machine Learning application, one has to determine first which degree of accuracy is needed relative to other factors. Time is a relevant factor to take into account. The time to print the coupons to customers, but also the time need for the algorithm to be 'trained' and ready.
In this case, a customer will generally not wait too long on the machine, therefore the algorithm should be able to provide the coupons within some seconds. 5 seconds is the limit I fixed for my project. In a more demanding context, 1 or 2 seconds would be the limit. The time for training the algorithm however is more flexible, on condition that the one-month period is not overtaken.
Another crucial point of this RS is the diversity of the recommendations.

Ideally, the machine would never print twice the same 8 coupons to two different customers. In reality this is unachievable, but the idea would be to have the largest diversity within the coupons. This constitutes the legitimacy for Machine Learning: changing the old marketing methods that focus on the product and advertising campains to targeting methods focusing directly on the customer. The more diverse the coupons, the more personalized the method.
Finally, the algorithm should be interpretable. If an error occurs it should be simple enough to give whys and the wherefores it happened, and not act like a black-box. It should also be scalable but here the dataset does not grow. On the long run however it may be necessary to study this point.

2.3 Datasets

Two datasets are available for this work:

- *loyalty card data*, which represents one month of purchases of customers of Retailer 1.
- *panel data*, which represents the history of purchases of households in Germany during one year.

These datasets are tables, where each row states for one purchase. Columns are relevant information about the purchase: customer ID, retailer ID (for the panel data), product ID, product category, product brand, quantity, value... Figure 2.3 and the next figures present in concrete the role of the RS: to predict whether the products have been purchased in another retailer than Retailer 1, then recommend these products. Then recommendating products consist only in selecting the products that have the highest probability.

Product	Label	Retailer
BUTTER	KERRYGOLD	2,3
FERTIGDESSERT	DANONE	1
KETCHUP	HEINZ	1
TAFELSCHOKOLADE	MILKA	2
VOLLWASCHMITTEL	SPEE	3
KETCHUP	JA !	
TAFELSCHOKOLADE	MILKA	3
VOLLWASCHMITTEL	PERSIL	
JOGHURT	MUELLER	
SPEISEEIS	LANGNESE	

Figure 2.3: Ideally we would know which products a customer buys and where

Product	Label	Retailer
BUTTER	KERRYGOLD	2,3
FERTIGDESSERT	DANONE	1
KETCHUP	HEINZ	1
TAFELSCHOKOLADE	MILKA	2
VOLLWASCHMITTEL	SPEE	3
KETCHUP	JA !	
TAFELSCHOKOLADE	MILKA	3
VOLLWASCHMITTEL	PERSIL	
JOGHURT	MUELLER	
SPEISEEIS	LANGNESE	

Figure 2.4: The recommendations we would make are shown by the green arrows

Product	Label	Retailer
BUTTER	KERRYGOLD	
FERTIGDESSERT	DANONE	1
KETCHUP	HEINZ	1
TAFELSCHOKOLADE	MILKA	
VOLLWASCHMITTEL	SPEE	
KETCHUP	JA !	
TAFELSCHOKOLADE	MILKA	
VOLLWASCHMITTEL	PERSIL	
JOGHURT	MUELLER	
SPEISEEIS	LANGNESE	

Figure 2.5: In the real world we only have access to the loyalty card data for a customer: which products have been bought in Retailer 1, and nothing else

Product	Label	P
BUTTER	KERRYGOLD	.80
FERTIGDESSERT	DANONE	
KETCHUP	HEINZ	
TAFELSCHOKOLADE	MILKA	.43
VOLLWASCHMITTEL	SPEE	.12
KETCHUP	JA !	.02
TAFELSCHOKOLADE	MILKA	.32
VOLLWASCHMITTEL	PERSIL	.89
JOGHURT	MUELLER	.45
SPEISEEIS	LANGNESE	.67

Figure 2.6: After predicting for each product the probability to be bought somewhere esle than Retailer 1, we recommend the ones with the highest probability

3 A Machine Learning Problem

The marketing problem is now translated to a concrete mathematical problem. This will constitute the cornerstone of the work,

3.1 Notations

In this part we define the mathematical notations required for implementing algorithms and for comparing them. While attributing symbols to the objects in the data, it seemed clear that I needed to make some changes in its form. If these modifications led to less information than there was originally in the dataset, one can still create features from the information that is lost and combine it with the new dataset to retrieve entirely the initial data.

The mathematical problem will derive from the following notations describing the data which contains:

- M different products available both in Retailer 1, and in all other retailers as a whole
- N households (or customers) from the panel data
- Y^1, and Y^{-1}, which denotes the $N \times M$ matrix of dummy variables $Y^1_{n,m}$, resp. $Y^{-1}_{n,m}$, which equals to 1 if the customer n has purchased the product m in Retailer 1, resp. not in Retailer 1, and zero else.
- N^X customers who have a loyalty card (to be correct, N^X is the number of different loyalty cards).
- X^1 is the $N^X \times M$ matrix of dummy variables $X^1_{n^X,m}$ which, as for Y^1, equals 1 if the customer who has the n^X-th loyalty card has purchased the product m in Retailer 1, zero else.

To recap, there are two matrices Y^1 and Y^{-1} from the panel data, and one matrix X^1 from the loyalty card data. Y^1 and X^1 are closely related since they both represent the purchases in Retailer 1. The idea is that predicting from X^1 is the same than predicting from Y^1. And the results of the latter are known, it is exactly Y^{-1}. Therefore, training our model on Y^1 and Y^{-1} should give a good prediction from X^1. Unfortunately there is no database (X^{-1}) of the true purchases of the customers who have a loyalty card. We discuss this issue within the section methodology.

3.2 Mathematical Problem

As developed previously, the challenge is to predict accurately the probability that a given customer purchases products in other supermarkets (than Retailer 1). With the preceding notations, we can pose the problem as follows.
For a customer $l \in [1 : N^X]$, written X_l as the l-th row of X, we are interested in evaluating the probability that this customer purchases the product $m \in [1 : M]$. The problem is then:

$$\mathrm{P}(X^{-1}_{l,m} = 1 | X^1_l, Y^1, Y^{-1}) \tag{3.1}$$

If we note $\mathscr{F} = (Y^1, Y^{-1})$, we can rewrite this formula by

$$\mathrm{P}(X^{-1}_{l,m} = 1 | X^1_l, \mathscr{F}) \tag{3.2}$$

This has to be done for every product $m \in [1 : M]$. We can write the estimation simultaneously as a vector formula:

$$f(X^1_l | \mathscr{F}) = \{\mathrm{P}(X^{-1}_{l,1} = 1 | X^1_l, \mathscr{F}), ..., \mathrm{P}(X^{-1}_{l,M} = 1 | X^1_l, \mathscr{F})\} \tag{3.3}$$

Thus the first recommendation for each customer l would simply reduce to the maximization program:

$$m^* = \underset{m \in [1:M], X^1_{l,m}=0}{\arg\max} \mathrm{P}(X^{-1}_{l,m} = 1 | X^1_l, \mathscr{F}) \tag{3.4}$$

Recommendations should only be made on products never bought in Retailer 1, that is why $X^1_{l,m} = 0$ in Equation 3.4. For p recommendations, we remove sequentially the maximum $p - 1$ times (8 in our case).

Finally, the coupon of 8 recommendations can be represented as $C(X_l^1|\mathscr{F}) = \{m_1, m_2, ..., m_8\}$, where

$$\forall m^* \in C(X_l^1|\mathscr{F}),\ \forall m \in \overline{C(X_l^1|\mathscr{F})} \cup \{\tilde{m},\ X_{l,\tilde{m}}^1 = 0\}$$

$$\mathrm{P}(X_{l,m^*}^{-1} = 1|X_l^1, \mathscr{F}) \geq \mathrm{P}(X_{l,m}^{-1} = 1|X_l^1, \mathscr{F}) \tag{3.5}$$

The notation $\mathscr{F}$ is chosen as to remind the "features". The set $\mathscr{F}$ can actually contain more information than Y^1 and Y^{-1} combined, for instance one could include the number of retailers each customer has been to, the fact that some products are milk products (product feature) ... The more features, the more likely the algorithm will be accurate. However it might also lead to overfitting but this is something we will discuss later.
Now that the problem is clearly posed, I can develop the methodology I use which is similar to the one we generally meet in other Machine Learning problems.

3.3 Methodology

3.3.1 Train/Test sets

As said previously, the dataset X^{-1}, which corresponds to the purchases of the customers having the loyalty card, is missing. Different techniques exist for this: the usual train/test sets separation, the K-fold cross-validation, or the train/validation/test sets separation and so on.
The choice made in this work is arbitrary if not convenient: I use the simple train/test cut, with 70% of the data kept in the training set and 30% in the test set. This is convenient due to the simplicity of the testing phase: only one run on the test set gives the results. Whereas the K-fold cross-validation for instance requires K times this test phase. The cut train/validation/test is often made when determining a parameter on the validation set, and then testing the method with the parameter on the test set. Here the methods do not correspond to this type of process, the simple train/test seems fair and convenient.
In order to retrieve the results, or compare the effiency of different algorithms, the train/test cut should be made pseudo-randomly. In `R` the command `set.seed()` should be placed before any further computation. Therefore

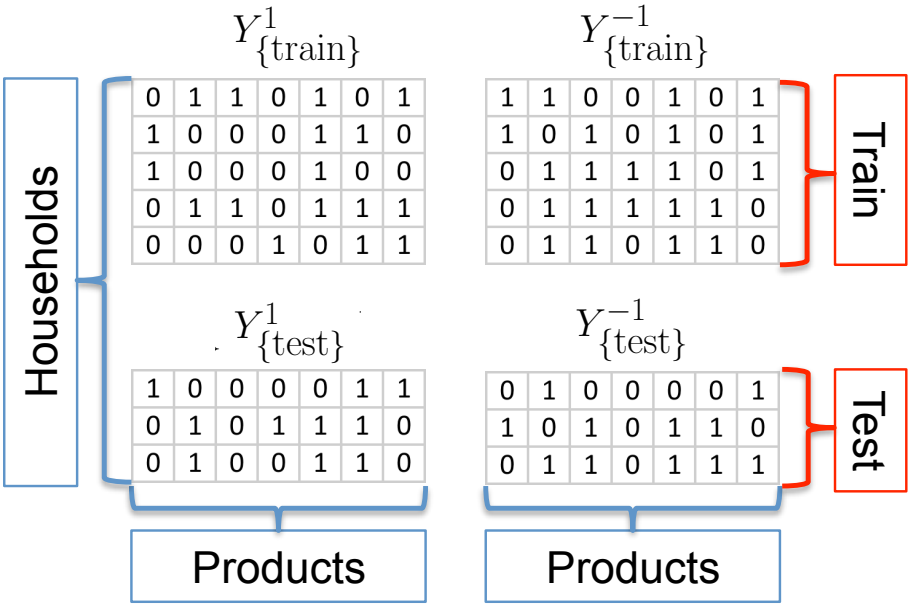


Figure 3.1: Illustration of the split of the initial dataset: 70% in the training set, 30% in the test set

we always get the same train/test separation for every algorithm and every rerun.

3.3.2 In Sample/Out of Sample error

When it comes to implement a statistical model to some data, the usual methodology is to build the model on the training set and evaluate it with the real values one should find. Here we compare the $\widehat{Y^{-1}_{\{train\}}}$ and $\widehat{Y^{1}_{\{test\}}}$. This raises the question of how to compare probabilities and dummy variables. Usually the difference to the bound 0.5 says whether it has been purchased (1) or not (0). But here the estimation step is only interesting for the first 8 products. A more relevant error mesure is needed.
If we use the previous notations, C which states for the coupon (the 8 rec-

ommendations) given to the customer X_l, we write:

$$\text{Rate of error} = 1 - \text{Rate of good guesses} \tag{3.6}$$

$$= 1 - \frac{1}{8}\sum_{m \in C} \mathbb{1}(X^{-1}_{l,m} = 1) \tag{3.7}$$

$$= \frac{1}{8}\sum_{m \in C} \mathbb{1}(X^{-1}_{l,m} = 0) \tag{3.8}$$

This formula suggests implicitly that if a product is recommended, it is expected to be purchased somewhere else (prediction on product m is counted as an error if $X^{-1}_{l,m} = 0$). Also, splitting the datasets into two sets leads to rewriting $\mathscr{F} = (Y^1_{\{train\}}, Y^{-1}_{\{train\}})$, with $C = C(\ .\ |\mathscr{F})$.
We therefore derive the in sample error:

$$\text{Rate of error IS} = \frac{1}{8}\sum_{m \in C} \mathbb{1}(Y^{-1}_{\{train\}\ l,m} = 0) \tag{3.9}$$

where $C = C(Y^1_{\{train\}\ l}|\mathscr{F})$; and the out of sample error:

$$\text{Rate of error OS} = \frac{1}{8}\sum_{m \in C} \mathbb{1}(Y^{-1}_{\{test\}\ l,m} = 0) \tag{3.10}$$

where $C = C(Y^1_{\{test\}\ l}|\mathscr{F})$. In some other research papers, the last error is called pseudo-out of sample because we still know its value, since $Y_{\{test\}}$ is in our data. The real out of sample error would be written with X, where we do not know the true purchases of the customers having the loyalty card.

3.3.3 Diversity measurement

Another aspect of the recommendations is their diversity. This point has been discussed previously and I define here the measure of diversity of an algorithm of recommendations. This is computed on both the training set and the test set. The notations define the measure for the training set but are also used on the test set.

$$\text{Rate of diversity} = \text{Proportion of different coupons} \tag{3.11}$$

$$= \frac{\text{Card}(C(Y^1_{\{train\}\ l}|\mathscr{F})^{n_{\{train\}}}_{l=1})}{n_{\{train\}}} \tag{3.12}$$

where the order within the 8 recommendations does not matter, i.e two coupons that are similar by permutation will not be counted twice in the cardinal.
Another way to measure the diversity is to count the number of times each product has been recommended. Meanwhile this cannot be seen as a single value, but as a array, and makes more sense when plotted on a chart.
These two measures are studied in detail in the section Results.

3.3.4 Naive methods

Of course if we implement some complex algorithms we expect good results. But is there a benchmark to say whether or not an algorithm is good or not ? It turns out that there are possible ways to select 8 products without using any statistical method.
The first idea one can have is to pick randomly for each customer 8 products that have not been purchased in Retailer 1. However this method fails since a customer usually purchases a slender minority of the total amount of products. The diversity on the contrary would be very high. But still, good accuracy prevails over good diversity.
Another idea would be to rank the products by the number of times they are purchased in all other retailers than Retailer 1. Then, according to the products a customer haven't purchased in Retailer 1, the recommendations consist in selecting the 8 best-selling products. This reasoning is intelligible as we face in this context a long tail phenomenon with the products. This means that only a few items make a majority of the sales. If this method can give good results, we see now that the diversity within the coupons should be very low, as only the best-selling products are recommended. I will call this strategy 'top8' from now on.
Again, another method would be to consider the best-selling products in Retailer 1 this time. It corresponds less to the idea of predicting the purchases in other retailers than the 'top8' strategy.
Naive methods can prove surprisingly good, and the aim would be to outperform these. The benchmark I will use for this work is the 'top8' strategy. As we will see in the next section, this strategy turned out to be relatively accurate.

4 Machine Learning Techniques

4.1 ABCs of Recommender Systems

A wide variety of Machine Learning techniques exists in the literature and proved efficient in many cases. But usually some methods work well in particular cases and not on other cases. Some other methods are versatile, however this ability to adapt often leads to less accuracy. Furthermore data cleaning is often responsible for having good results whatever model is used. The step of preprocessing is paramount for getting the expected accuracy. Whenever Machine Learning seems a good solution to a problem, one has to be careful to use the good methods with a precise prior preprocessing step. To enlarge the debate on the comparison between models, [4] unveils a sensible aspect in Machine Learning: a complex model does not necessarily outperform a simple one.
First I will describe more in depth what researchers understand by the term Recommender System, where I will quote frequently [1]. The RS is usually one of the four branches in the field of Machine Learning. The three other branches are Supervised Learning (SL), Unsupervised Learning (UL), and Reinforcement Learning (RL). RS is particular in the way that it is a combination of these previous aspects. People may have ordered preferences within a set of products, but it is very unlikely that it will be exactly known to anyone. Therefore the RS tries to estimate the preferences, which exist, but the genuine set of ordered preferences will never be known. RS locates simply between SL and UL. Reinforcement Learning can also be used as there are often explicit feedbacks from the recommended products (rating, comments...) or implicit ones (purchase, click...).
In practice, there are many different ways of recommending products to customers. A non exhaustive list of methods is presented below:

- Collaborative Filtering, which uses the customers past behavior. This has often been the first choice for forecasting. There are two different methods for making recommendations:
  - User-based, which focuses on looking for users who have a similar basket of purchases. This can be seen easily with a matrix having customers as rows and products as columns. With this representation, this method looks for similarities between the rows (customers' baskets).
  - Product-based, focuses on looking for products which have similar pattern in their sales history. With the previous illustration, this method looks for similarities between the columns (products' sales history).
- Demographic, which puts the interest on the customer features. For instance the age, the sex, or the affinity to bio products. This method clusters customers in order to make specific recommendations within each cluster.
- Content-based, which puts the interest on the product features. For instance if a product is dairy, for diet, or is bio. This method clusters products in order to make recommendations for one or a small number of clusters of products.
- Social, which are connected to the customers' social circle's purchases.

These methods lie on particular databases. For this work the available data would push to use the Collaborative Filtering (CF) method. If I had more time, another way would be to build a matrix of products' features, by creating manually features like "dairy" or "quantity of vitamin C" and filling out for each product the corresponding value. Improving the database, adding more features to it can help to get better results for instance. Since the one-month period is already short, I did not address this issue and went straightforward on the CF methods. With the experience of the Netflix prize a large number of CF methods are now popular and I will focus on a subset of them.
The goal here is to estimate accurately the vector of probabilities to purchase a product not in Retailer 1. These probabilities should be computed in the same step of the process otherwise both the complexity will be to high and

the probabilities may not be comparable since the method to compute them is different. Thus the wish is to find methods that can compute similarly the probabilities, and being as personalized as possible.

4.2 Naive Bayes

The Naive Bayes method as other widespread methods, uses the famous Bayes formula:

$$\mathrm{P}(A|B) = \frac{\mathrm{P}(B|A)\,\mathrm{P}(A)}{\mathrm{P}(B)} \tag{4.1}$$

With the notations developed above, the previous formula becomes for a customer $n \in [1 : N]$ and a product $m \in [1 : M]$, the probability that a customer, represented by X^1, purchases product m not in Retailer 1 is

$$\mathrm{P}(X_m^{-1} = 1|X^1) = \frac{\mathrm{P}(X^1|X_m^{-1} = 1)\,\mathrm{P}(X_m^{-1} = 1)}{\mathrm{P}(X^1)}$$

and similarly the probability that this customer does not purchase product m not in Retailer 1

$$\mathrm{P}(X_m^{-1} = 0|X^1) = \frac{\mathrm{P}(X^1|X_m^{-1} = 0)\,\mathrm{P}(X_m^{-1} = 0)}{\mathrm{P}(X^1)}$$

where in testing on the data, X^1 is simply a row of Y^1 and X^{-1}, the unknown vector of purchases, is the row corresponding to the same customer in Y^{-1}. I will continue using the notations Y^1 and Y^{-1} instead of X^1 and X^{-1} to illustrate my real implementation. Therefore $Y_n^1 = X^1$ and $Y_{n,m}^{-1} = X_m^{-1}$.
The Naive Bayes is a particular method which supposes that every feature is independent of the others (which is why it is called 'naive'). The tricky part of the calculation now uses the term $\mathrm{P}(Y_n^1|Y_{n,m}^{-1} = k)$ for $k \in \{0, 1\}$ in the previous equation. One can rewrite this term with respect to the independence condition for $k \in \{0, 1\}$:

$$\mathrm{P}(Y_n^1|Y_{n,m}^{-1} = k) = \prod_{l=1}^{M} \mathrm{P}(Y_{n,l}^1|Y_{n,m}^{-1} = k) \tag{4.2}$$

which means that the probability that a customer has a certain behavior in Retailer 1 knowing if he purchases or not product m in another retailer, is

simply the product of the probabilities of each individual behavior on the products in Retailer 1. Then, each term of the product is a simple count in the matrices. For $(m, l) \in [1 : M]^2, n \in [1 : N]$,

$$\mathrm{P}(Y^1_{n,l}|Y^{-1}_{n,m} = k) = \frac{1}{N}\sum_{n'=1}^{N} \mathbf{I}(Y^1_{n',l} = Y^1_{n,l}|Y^{-1}_{n',m} = k) \tag{4.3}$$

Now in order to compare $\mathrm{P}(Y^{-1}_{n,m} = 1|Y^1_n)$ and $\mathrm{P}(Y^{-1}_{n,m} = 0|Y^1_n)$, we take the logarithm of the ratio which is called δ_m:

$$\begin{aligned}
\delta_m &= \log\left\{\frac{\mathrm{P}(Y^{-1}_{n,m} = 1|Y^1_n)}{\mathrm{P}(Y^{-1}_{n,m} = 0|Y^1_n)}\right\} \\
&= \log\left\{\frac{\mathrm{P}(Y^1_n|Y^{-1}_{n,m} = 1) \times \mathrm{P}(Y^{-1}_{n,m} = 1)}{\mathrm{P}(Y^1_n|Y^{-1}_{n,m} = 0) \times \mathrm{P}(Y^{-1}_{n,m} = 0)}\right\} \\
&= \sum_{l=1}^{M} \log\left\{\frac{\mathrm{P}(Y^1_{n,l}|Y^{-1}_{n,m} = 1)}{\mathrm{P}(Y^1_{n,l}|Y^{-1}_{n,m} = 0)}\right\} + \log\left\{\frac{\mathrm{P}(Y^{-1}_{n,m} = 1)}{\mathrm{P}(Y^{-1}_{n,m} = 0)}\right\}
\end{aligned}$$

After obtaining the vector $(\delta_1, \delta_2, ..., \delta_M)$ for a customer n, the remaining task is to get the 8 highest δ_m, knowing that $Y^1_{n,m} = 0$ of course.

4.3 Linear Discriminent Analysis

The LDA method do not uses the assumption of independence. It supposes however another hypothesis, which is that the probability of both purchasing product m not in Retailer 1, and having a specific behavior in Retailer 1 is gaussian. In mathematical terms, it gives for $k \in \{0, 1\}$:

$$\mathrm{P}(X^{-1}_m = k|X^1) = \frac{\pi_k f_k(x)}{\pi_1 f_1(x) + \pi_0 f_0(x)}$$

where

- $f_0(x) = \mathrm{P}(X^1 = x|X^{-1}_m = 0)$ the multivariate density of X^1 knowing $X^{-1}_m = 0$.
- $f_1(x) = \mathrm{P}(X^1 = x|X^{-1}_m = 1)$ the multivariate density of X^1 knowing $X^{-1}_m = 1$.

- $\pi_0 = \mathrm{P}(X_m^{-1} = 0)$ the prior probability for $X_m^{-1} = 0$
- $\pi_1 = \mathrm{P}(X_m^{-1} = 1)$ the prior probability for $X_m^{-1} = 1$

and

$$f_k(x) = \frac{1}{(2\pi)^{M/2}|\Sigma_m|^{0.5}} exp - \frac{1}{2}(x - \mu_{m,k})^\top \Sigma_m^{-1}(x - \mu_{m,k})$$

This formula is at first sight quite complex, but it simply states that given a product m, the probability to purchase it or not somewhere else than in Retailer 1, is of a gaussian form.
Once the parameters estimated $(\mu_{m,k}, \Sigma_m)$ the probabilities can be computed, and the linear discriminent δ_m is computed as follows:

$$\begin{aligned}\delta_m &= \log\{\mathrm{P}(X_m^{-1} = k|X^1)\} + C \\ &= X^{1\ \top}\Sigma_m^{-1}\mu_{m,k} - \frac{1}{2}\mu_{m,k}^\top \Sigma_m^{-1}\mu_{m,k} + \log(\pi_k)\end{aligned}$$

It is clear that in the context of this work there are too many parameters to estimate: 2 parameters for each product, a model which will massively overfit since there are $\sim$2000 customers for $\sim$4000 products.
One possible way to adapt this method would be to reduce significantly the number of products to lower the number of parameters. But this still does not solve the problem because it would give probabilities for a limited amount of products.

4.4 k-Nearest Neighbors

This method is directly taken from the field of non-parametric regressions. As simple as it is, it often surprises with good results. The core of the method lies on the fact that some points in the data have more relevance than others to predict for a certain X. This uses a very simple idea similar to the one of the histogram: group the data in bins and predict by averaging over these bins. The bins are here neighborhoods of size k. This means that for a given X, one has to find a neighborhood of X in the data, and predict the class (purchase or not purchase) that is dominant in the neighborhood.
In practice, the probability to purchase product m is simply the sample

probability on the neighborhood. Therefore, if the k nearest neighbors of the customer X are $n_1, n_2, ..., n_k$, then

$$\mathrm{P}(X_m^{-1} = 1|X^1) = \frac{\sum_{l=1}^{k} \mathbf{I}(Y_{n_l,m}^{-1} = 1)}{k}$$

This formula is really straightforward. The caveat is to construct a good way to find these k neighbors, and eventually to choose a good k, but the last point is easier.
For continuous variables, the usual similarity functions are the sample correlation, the Kendall's rank correlation and so on. When the variables are discrete, the similarity function is much more delicate to build. The functions cited above usually do not work very well.
Once the similarity function selected, the way to find k is to use the so-called cross-validation method.
This method is very attractive for practitioners for both being super fast and for proving quite accurate, if the similarity function is chosen smartly.
It is also possible to use the kNN method on the products side. The same issue arises concerning the choice of the similarity function between products.

4.5 Decision Tree

Decision trees are very popular in applications where interpretability is paramount. For medical purposes for instance. In our case, this method could be used to give probability predictions. If one small tree is created for each product, then this method could provide very quickly a vector of probabilities, subject to having made the trees which may take some time.
Usually a tree is defined by decision nodes until reaching end nodes. The aim is to build short trees for predicting the probability $\mathrm{P}(X_m^{-1} = 1|X^1)$ quickly. The main problem that this method faces is that each tree can have different accuracy, and it might turn out that the recommendations we make rely on poor predictions.
Another way of using this method could be to build a large tree which provides at each end node the vector of probability. But if we suppose the tree has C end nodes, only C different coupons will ever be given to customers. Therefore, to provide both accuracy and diversity, a tree must be sufficiently large and might overfit.

4.6 Multivariate Probit

The multivariate probit model uses the fact that the observed purchased are binary. Like the LDA method, it has several parameters to estimate. The explanatory variable would be the purchases in Retailer 1: $X_1^1, X_2^1, ..., X_M^1$, and the explained variables are the latent variables ${X_1^{-1}}^*, {X_2^{-1}}^*, ..., {X_M^{-1}}^*$ where ${X_m^{-1}}^* > 0 \Rightarrow X_m^{-1} = 1$. Then the model is for $m \in [1 : M]$:

$$ {X_m^{-1}}^* = X_m^1 \beta_m + \varepsilon_m \tag{4.4} $$

where

$$ (\varepsilon_1, ..., \varepsilon_M)^\top | (X_1^1, ..., X_M^1)^\top \overset{\text{as.}}{\sim} N(0_M, \Sigma) \tag{4.5} $$

with 0_M is the null vector of length M, and Σ the covariance matrix of $(X_1^1, ..., X_M^1)^\top$.
This model then is estimated by maximum likelihood. Most developed softwares will provide a built-in likelihood maximization program.
This model turns out to be a simple gaussian copula model, where the copula of a M variables $X_1, ..., X_M$ with their marginal distributions $F_1, ..., F_M$ is

$$ C(F_1(x_1), ..., F_M(x_M)) = \mathrm{P}(X_1 \leq x_1, ..., X_M \leq x_M)) \tag{4.6} $$

Other models with more complex copula could be considered.

4.7 Methods for clustering

These methods might help to reduce the complexity of the problem. They divide the initial dataset into several clusters, which should generally represent the data correctly. These clustering methods might be used on the customers as well as on the products. Usually one has to have some features to back these clustering methods. A feature does not necesarily stand for the data one wants to predict. For instance, it might be relevant to use the fact that products are "dairy" to cluster them, even if this has no role in the final probability computation.
This problem is based on estimating a vector of probabilities of length M. If one wishes to use a general clustering method to divide the data into the number of different classes, an issue is looming behind: the number of different classes. For the M products, the classes are $\{0, 1\}^M$, therefore 2^M

classes. This number is way larger than the number of customers. Thus a clustering method should not intend to separate the classes, but help to get some structure in the data.
The last remark to make is that one of the aim of the RS in this case is to have the create the most personalized recommendations. In other terms, the coupons would ideally be different from one customer to another. Or mapping the data with clusters should gather customers in ensembles. If the recommendations are made from these clusters, then there is a real lack of diversity in the coupons. Therefore if one clustering method is used, then another method should be applied on top of the former to make the recommendations unique.

4.7.1 K-means

The K-Means algorithm is quite famous in the field of Unsupervised Learning, since it is one of the fastest way to compute clusters in the data. The principle of this method is once the number of clusters is set up, K, the algorithm builds sequentially the K clusters parallel to the K centers until convergence.
This method may be very simple to compute, yet the underlying motivation for choosing K is less clear. For one K the clustering might work well although for another K it might perform poorly. Even for the same K, if the centers are chosen randomly, there is almost no chance to retrieve the exact same results twice. To make a good use of this algorithm requires many tries, which lessens the fact it is very fast.

4.7.2 Support Vector Machine

This method is strongly supported in many papers and proved very effective in some cases. This is also a clustering method but in Supervised Learning, meaning that the classes are known. It maps the data like a web which tends to divide the classes with the largest margin between them.
The issue is that there are 2^M classes in the problem. Therefore this method cannot pretend to represent every class individually. How to map the data and give probabilities for each purchase in the same time ? It seems that there is no such clever way to do that.
Consequently this method should only be used to separate the data into sev-

eral clusters that have to be chosen in a smart way. For instance, separating the customers who purchase the most to the customers who purchase the less. Or grouping together the ones who purchase specific products in Retailer 1, and so on.
If used, this method should perform many blind attempts and it might take a lot of time before finding a smart separation.

4.8 Comparing the methods

This plethora of methods provides a real variety of statistical techniques. Some are more focused on an implicit model, like LDA or Multivariate Probit, and some other lie only on the data, like kNN or NB. The table below underlines the main differences.

	Adjusted for vectors	Not too many parameters	Low complexity	Has some degree of freedom	Adapted for highly multivariate
kNN					
LDA					
Tree					
NB					
SVM					
K-M					
Probit					

- *Adjusted for vectors* is for outputting a vector of estimated probabilities
- *Not too many parameters* is simply to avoid the estimation errors
- *Low complexity* to train and to test
- *Has some degree of freedom* means if once the algorithm produces the coupon, are there ways to improve it without changing completely the algorithm and not touching the data ?
- *Adapted for highly multivariate* points out that the algorithm should give relevant results even if there are more products than customers

The kNN, the Naive Bayes and the K-Means stand as the best candidates for the next section.

5 Implementation and Results

5.1 Data Analysis and Preprocessing

In this part are described the different preprocessing steps induced by a data analysis. As the dataset generated by the loyalty card will only be kept once the algorithm is built, the terms 'customer' and 'household' will both state for the same unique entity: the people from the panel data. This will remain true until said otherwise.
In the section 3.1 I used the general matrix notation, with customers as the rows and products as the columns. Implicitly, this notation surmised that the products offered by the retailers were the same. In practice this is not true. Actually there are more than ten thousand different products ID in the dataset. It is interesting to plot the revenue generated by the sales for each product, as shown on Figure 5.1. A first rational thing to do is to remove this very long tail, which denotes seasonal products, special offers or simply scarce products sold in a minority of retailers. Just before removing this heavy tail it is interesting to look at the retailers revenues which also display a long tail phenomenon, cf Figure 5.2. The discrepancy in the retailers revenues is also due to outliers which are online retailers, retailers which are not primarily focused on consumer goods related to food and so on. Another step of preprocessing is thus needed.

Also, encouraged by the company which provided the data, I developed a special taxonomy of the products, namely the 'recommender ID', which is the fusion of the product category and the brand. The idea is that different product IDs state for the same product category and the same brand. For example, if a customer has purchased a small bottle of soda in Retailer 1, there is a good chance that he knows the large bottle of soda of the same brand. Therefore there is no value added by recommending the large bottle since the customer could have already thought about buying it. Moreover

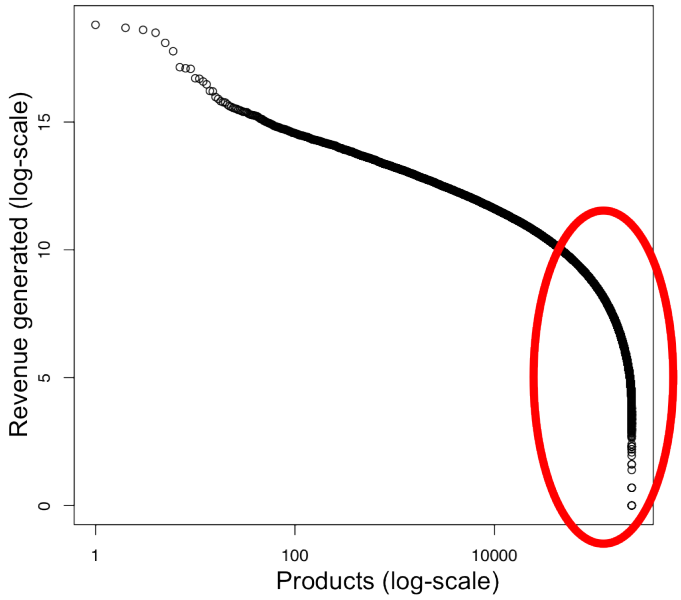


Figure 5.1: Revenue generated per product on a log-log scale. The red oval stresses the very long tail existing in the set of products

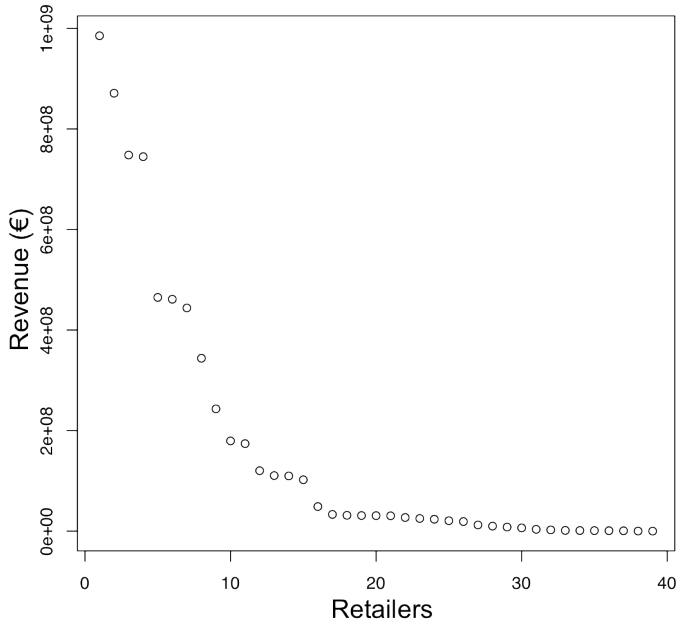


Figure 5.2: Revenue generated per retailer

there if a coupon suggests both the small and the large bottle as two distinct recommendations, it maybe would have been better to simply recommend once the bottle (small or large) and another product. That is why the fusion of the product category (in the example: bottle of soda) with the brand seems to define more clearly the recommendations. From now on, the term 'product' will lose its connotation of size or format and will simply refer to the term 'recommender ID'.
Then, another necessary preprocessing is carried out: keeping only products available both in Retailer 1 and all other retailers as a whole, and selecting only customers who have purchased at least once in Retailer 1. This keeps the households who are customers of Retailer 1, even not regular ones, and helps to understand the different behaviors when it comes to Retailer 1 and all other retailers.
A plot of the revenues generated by the products in both Retailer 1 and all other retailers is shown on Figures 5.3 and 5.4. From these charts and particularly on the first one a famous law of large number appears: the Zipf's law, where the log-log scale reveals a liner relationship. Indeed, this law was originally discovered by Zipf when looking at the occurencies of every word in the book *Ulysse* by James Joyce. Here the words are replaced by the products, the occurrences by the sales revenue, the book is replaced in the first chart by Retailer 1, in the second by all other retailers aggregated. Therefore it makes sense to notice almost a straight line on the first chart, since one retailer can be assimilated to one book, and a curved line on the second chart since aggregating retailers that are different will certainly not manifest an homogeneity in the set of products ranked by sales revenue.

These steps of preprocessing removed the possible outliers in the products, the retailers and the customers. The long tail phenomenon, present in these three entities, induces outliers but generally the long tail is to be taken into account in the data. There will be a tough choice to make between keeping the long tail and both a good accuracy and a fast algorithm.
It is still interesting to look at what remains the data after this severe cleaning process. The customers are the ones who have purchased at least once in Retailer 1, the products are the 'recommender IDs' which are common to both Retailer 1 and all other retailers as a whole, and the retailers are the acutal competitors of Retailer 1. The latter is a Retailer of medium size and faces cutthroat competition with hard discount retailers which gather most of the household's expenses.
In order to grasp the new datasets, the Figures 5.5 and 5.6 display respec-

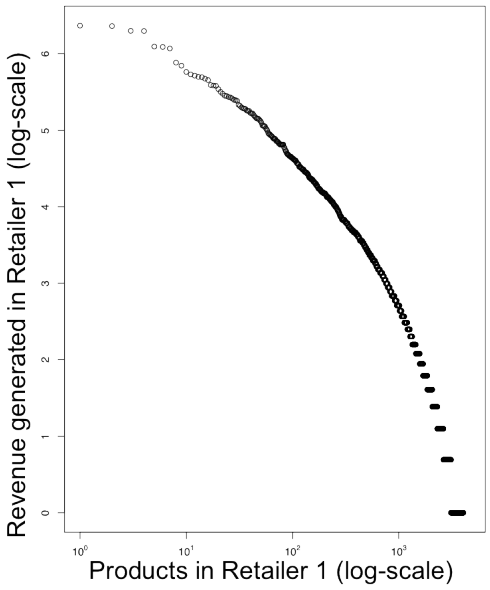


Figure 5.3: Revenue generated per product on a log-log scale in Retailer 1

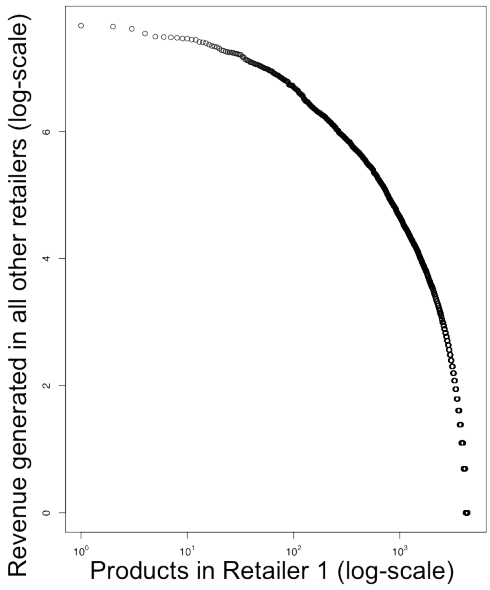


Figure 5.4: Revenue generated per product on a log-log scale in all other retailers

tively the number of different products bought in Retailer per customer and the sales revenue per product also in Retailer 1. Both a red and a grey line are plotted on these charts and stand for respectively the 50% tail and the 90% tail. Is is noticeable that the long tails are still strong on both the customers side and on the products side. This may pose problems for the algorithm since the long tail might give less relevant information and confuse the logic within the algorithm.

To complete this priliminary data preprocessing, it is necessary to know that the notations developped in section 3.1 are in line with the new data. Actually the data has never been transformed into matrices. It is sufficient to build two contigency tables, one for the purchases in Retailer 1, one for the purchases everywhere else.
However we see that there is a huge loss of information from transforming the contigency tables into matrices of dummies: the quantity is lost and becomes a boolean variable. If there was more time for the project I would have focused on trying methods using the quantity. But the matrix notations requires some consideration. In fact, this notation reveals the purchases of

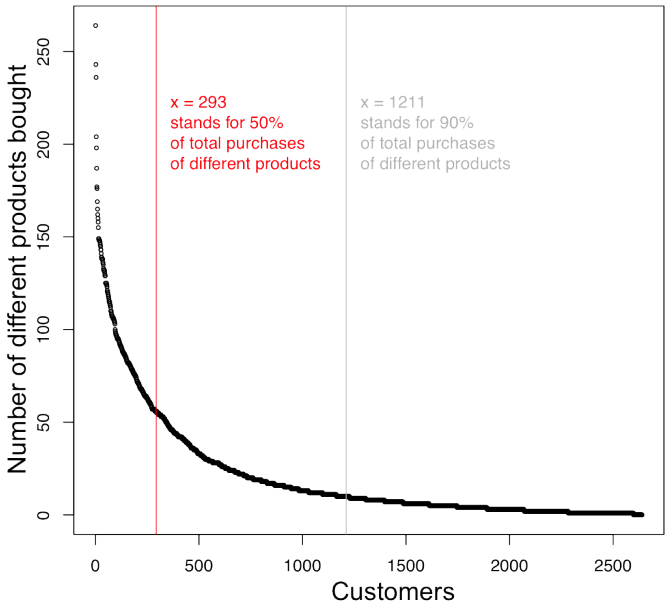


Figure 5.5: Number of different products bought in total in Retailer 1 per customer

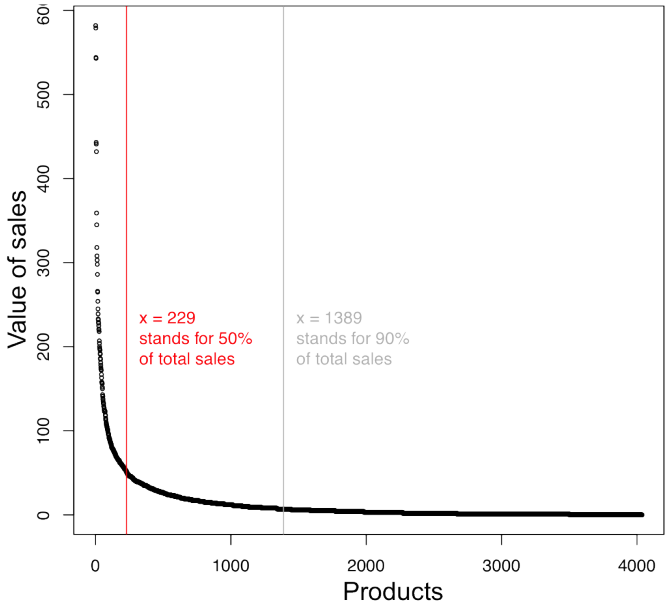


Figure 5.6: Sales revenue per product in Retailer 1

the customers, but also stresses the non-purchases (the zero values in the matrix).
This 'new' information, the absence of a purchase, is actually wider than the previous information, the presence of a purchase. This fills out the matrices with a raft of zeros and leads to very sparse data which constitutes a serious issue. The reason of the concern is that one cannot say whether an absence of purchase means that the customer does not like the product, or that he did not have the chance to buy it.
The customer may not have bought a product even if the products correspond to his tastes and budget, because of multiple possible reasons: the timeframe (one year in this case) is too small to capture this purchase, the customer did not send this information to the data provider (incomplete data collection), or simply because all the retailers other than Retailer 1 display different products and the set of products available in Retailer 1 is not entirely available in each other retailer individually.
One of the challenge of this Machine Learning problem is to understand to which point this sparsity will disturb the algorithm, and then to choose the methods which suffer the less to this sparsity issue. Indeed, sparsity is met in many other cases, for instance in [1]. I made the choice to not restrict the selection of algorithms to the resilience of sparsity. Instead, I let the tail factor the possibility to varry in order to compare the performences of each algorithm in different situations of sparsity.
To grasp this issue, the Figures 5.7 and 5.8 display the sparsity of respectively Y^1 and Y^{-1}. The tail factor t_{cust} accounts for the customers' number of different purchases in Retailer 1 while the tail factor t_{prod} accounts for the products' sales revenue both in Retailer 1. Removing the tail in Retailer 1 does not necessarily remove the tail in the other retailers (Y^{-1}).

5.2 Results

5.2.1 Preliminary

Out of all the possible choices of Machine Learning methods, 4 were actually tested. The implementation of the 'top8' strategy, which constitutes a RS, has also to be added up.
The methods I used, the Naïve Bayes, the K-Means and two types of k-Nearest Neaighbors, were first tested on the training set and, appart from

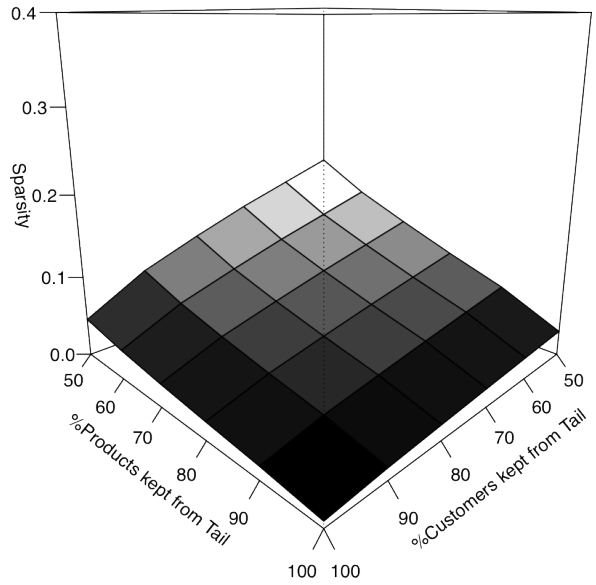


Figure 5.7: Sparsity with Y^1 relative to the proportion of the tails to keep in the sets of customers and products both of Retailer 1

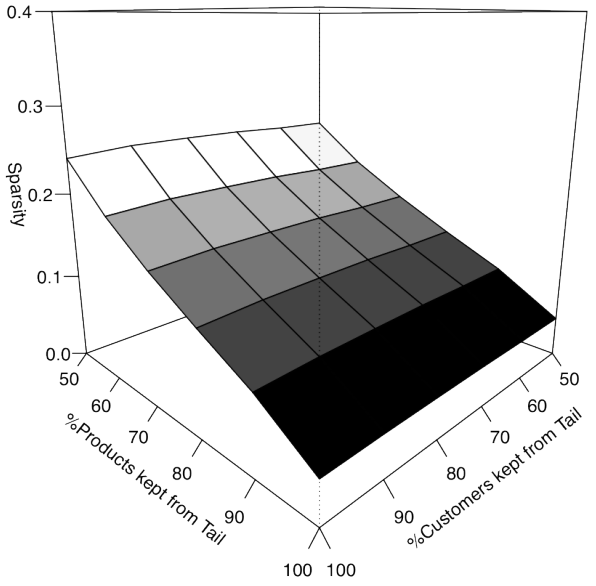


Figure 5.8: Sparsity with Y^{-1} relative to the percentage of the tails to keep in the sets of customers and products both of Retailer 1

the K-Means, were then implemented on the test set with prior discussion and final choice for the tail factor concerning the products. Regarding the tail factor on the customers, it seemed not adequate to make it different than 100%. Indeed, the algorithm should use all the available data of customers since the number of customers is the sample size. On the contrary the number of products could be seen as the number of variates within the problem and can or should be reduced in size to gain accuracy.
A caveat on the implementation side: the algorithms were written in `R` and without the help of other packages (appart form K-Means). This implies that more time was spent to check to confirm the validity of the algorithms. At the same time it allowed me to implement several changes easily, for instance for the distance function within the k-Nearest Neighbors method.

5.2.2 How to read the charts ?

The results are displayed as tables. The algorithms were tested on the training set while changing the two tail factors (t_{cust} and t_{prod}) from 50% to 100% with 10% increment each time. In total, $6 \times 6 = 36$ tries were made, that is why the tables are represented as 6×6 matrices. Keeping the tail as a tuning parameter helps to understand the efficiency of the method when the sparsity of the data varies.
Figures are shown using the well-known contour plot, which helps visualizing data of three dimensions in two dimensions. The color red indicates a low number while light yellow stands for high numbers. The figures plotting the accuracy target the light yellow areas (good predictions), while figures displaying the run-time complexity target the red areas (low complexity). Moreover, the diversity as the proportion of unique coupons would also focus on the light yellow areas (high diversity).
The units are for the first and third charts ratios, from 0 to 1 (they both denote a rate), and the seconds for the second chart. A black digit indicates an average value for the colored areas on the accuracy charts.

5.2.3 In sample Results

top8 - recommending the best-selling products

This 'top8' strategy, which is the benchmark that the Machine Learning algorithms should beat, is fairly simple to code. The idea is to sort the num-

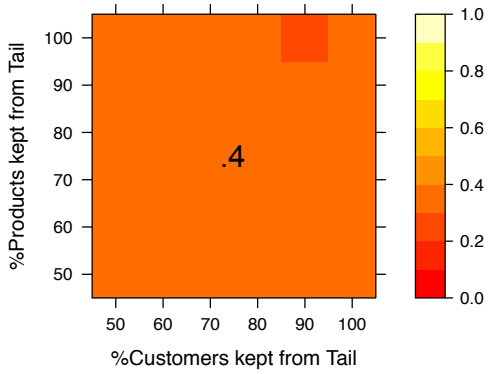


Figure 5.9: Rate of good predictions top8 algorithm

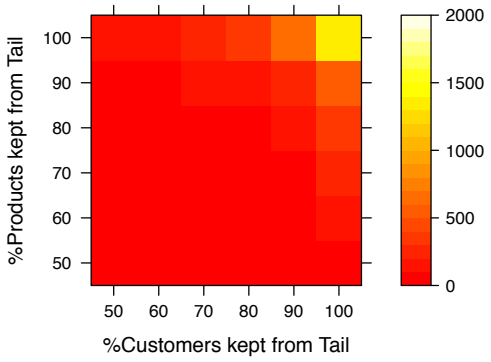


Figure 5.10: Run-time complexity top8 algorithm

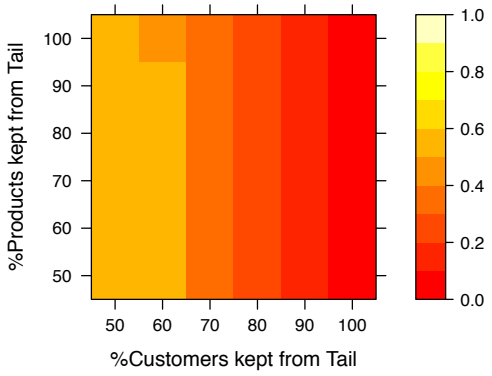


Figure 5.11: Diversity of coupons top8 algorithm

ber of sales per product in all other retailers than Retailer 1 in decreasing order, and recommend the best-selling products which have not been purchased by the customer in Retailer 1.

This algorithm is very fast, and its complexity serves as a lower bound for the other algorithms. On the contrary, its accuracy and its diversity are to be challenged by the other methods.

Looking at the results of this algorithm leads to several interesting points:

- whatever the tail factors, the accuracy remains stable, between 40 and 50%. This is quite impressive since there are more than 4000 products. This method actually performs very well and might prove difficult to beat.
- the diversity decreases with the number of customers, which is quite straightforward since the coupons are more and more similar since the best-selling products won't change.

If this method turns out to be very accurate, the number of different coupons given is actually very low thus not personalized at all, as expected.

k-Neareast Neighbors

Many packages provide k-Nearest Neighbors algorithms in `R` (the basic `class` package for instance). However, a thorough analysis would tend to rely on a user-made kNN. Datasets vary significantly from one application to another, and the performance of the kNN may be great in one case and terrible in another one.
By making my own kNN, I could input different distance functions between the customers. Actually the two kNN (namely $k\text{NN}^1$ and $k\text{NN}^2$) I implemented rely precisely on two different distance functions (or similarity function) since they use the same $k = 100$.
Finding the optimal number of neighbors, k, can prove very hard using cross-validation when large datasets are involved. Therefore I used the k that was advised by the company providing the data: $k = 100$.
The first algorithm uses the basic sample correlation as the distance function. Of course for binary variables it might not be the best choice for a distance function. Yet as a first try it is a reasonable choice since it is very fast to

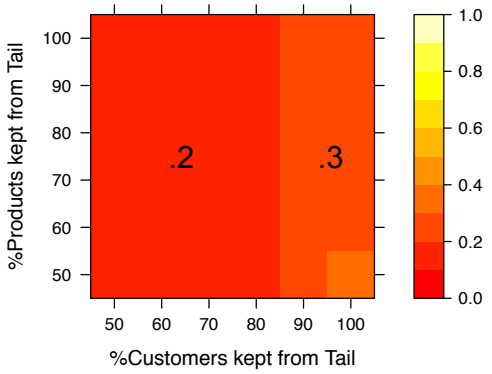


Figure 5.12: Rate of good predictions $k\text{NN}^1$ algorithm

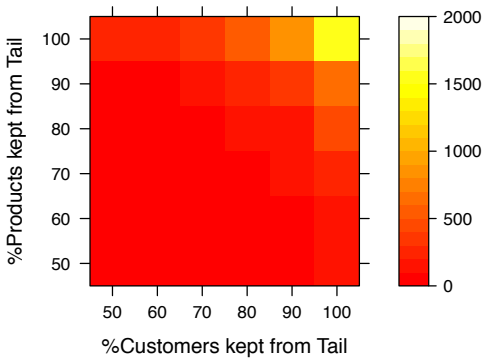


Figure 5.13: Run-time complexity $k\text{NN}^1$ algorithm

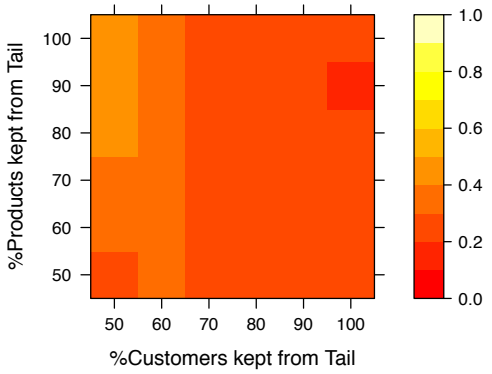


Figure 5.14: Diversity of coupons $k\text{NN}^1$ algorithm

compute and the correlation still makes sense even if it does not seem to be the best choice.
The results from the kNN[1] algorithm arouse different comments:

- the performance is worst than the 'top8' at around 30%, which is true everywhere on the 6×6 table. This suggest that the model is not good enough.
- the performance increases with the number of customers, which points out that actually the more data, the more accurate the algorithm. This is an example of a high-variance algorithm where data have a strong impact on the performance.
- the diversity decreases as the number of customers increases, which is the same effect when looking at the results of the 'top8' algorithm and does not surprise.

There was therefore some room for another implementation of the kNN algorithm. Since the performance was poor, two possibilities were offered: to change the number of neighbors k, or to change the distance function, here the sample correlation function. As said above, the distance function is not appropriate to binary variables. Furthermore, it seems that in the way it is built, this kNN[1] finds similarity between customers giving the same weights to both 1 and 0. As the 0 is dominant in the data, this algorithm simply cluster customers by what they did not purchase.
At this point it is relevant to try to build a distance that would focus mainly on the 1 in order to group customers by what they purchased, which is the most interesting information after all.
A simple distance would then, for a given customer, compute the similarity of the other customers on the products bought by the former. In practice, the distance function (which is not really a distance since $d(A, B) \neq d(B, A)$), or the similarity function, counts the number of common products bought with a customer. The customers that have the highest number of common products purchased with the reference customer are then his neighbors.
This similarity function led to the kNN[2] algorithm, which had particularly impressive results:

- the kNN[2] outperforms in accuracy every other algorithm, especially the 'top8' benchmark. This remarkably uses the fact that there is much

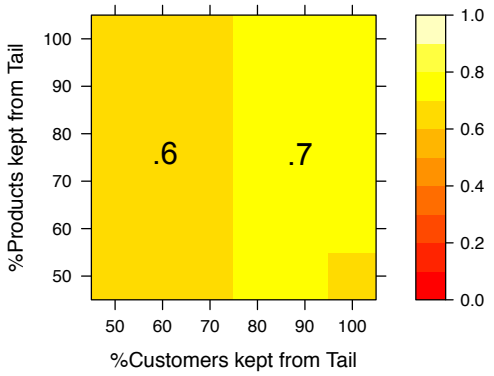


Figure 5.15: Rate of good predictions kNN^2 algorithm

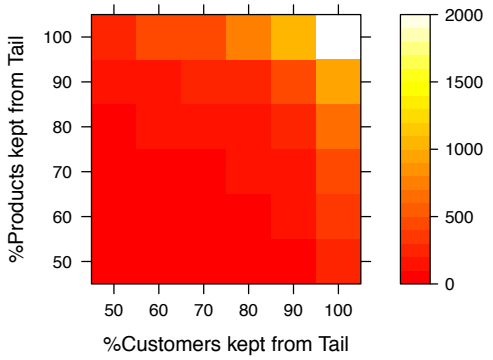


Figure 5.16: Run-time complexity kNN^2 algorithm

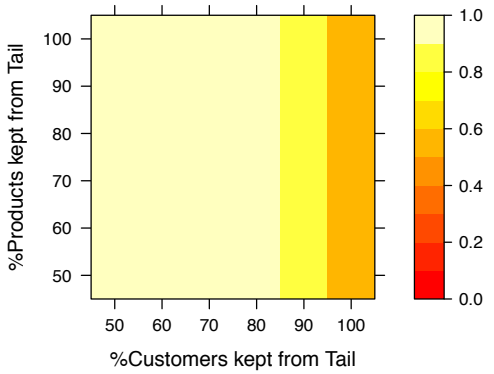


Figure 5.17: Diversity of coupons kNN^2 algorithm

more value in looking for basket similarity between customers than the similarity between whole set of products which kNN[1] does.

- the performance increases with the number of customers, which is the same remark from the kNN[1]. The fact that an algorithm performs better than another does not change its high-variance attribute: more data leads to better accuracy.

- the diversity drops when the customers of the very long tail are present. This is without doubt due to the fact that the customers from the long tail did not bought much products, leading to not very accurate neighborhood borders. And to the fact that these customers might purchase the same kind of products, for instance staple products as they are not regular customers of Retailer 1, which induce the same coupons to be printed.

- the run-time complexity rose compared to the two previous algorithms, but remain under a reasonable bound.

Naive Bayes

The Naive Bayes algorithm was built from scratch as explained in the section 4. Its run-time complexity spirals as the tail factors increase. This is one of the main problem with the NB algorithm.
When tried in-sample, the NB turned out to take so much time to run that I significantly reduced the tries. Only a sample of the complete 6×6 output matrix were computed. Only 6 algorithms were tried (actually one algorithm was tried on 6 different datasets, different by the tail factors).
These results reveal different points:

- time complexity is very high (the legend is different from the previous charts): indeed, the computation is $\mathcal{O}(N_{\{train\}} \times M)$ for a single prediction, using the same notations described in 4. Therefore this in-sample error takes $\mathcal{O}(N^2_{\{train\}} \times M)$ to compute.

- when t_{prod} is low, the accuracy is good. This is very interesting, since it suggests that the method works only for products with a significant number of 1. Actually, it is not the number of 1 which matters but the balance between 1 and 0. When the information is well-balanced, or at least not tipped to one side, then the probabilities are computed

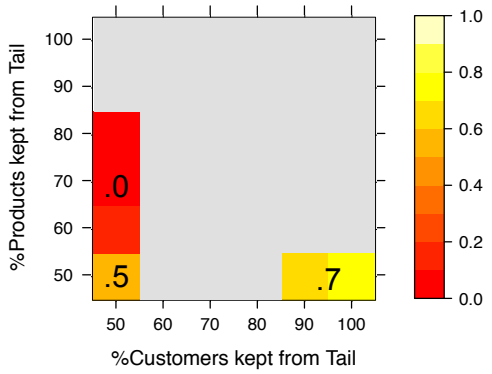


Figure 5.18: Rate of good predictions Naive Bayes algorithm

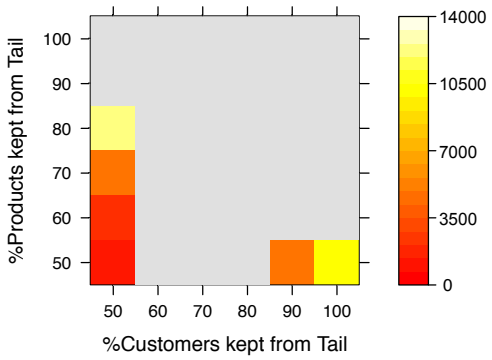


Figure 5.19: Run-time complexity Naive Bayes algorithm

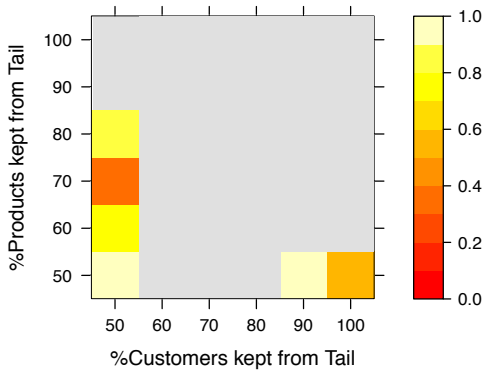


Figure 5.20: Diversity of coupons Naive Bayes algorithm

on reasonable sample sizes and are therefore pertinent. No conditional probability is then inclined to be near 0 and unbalance the overall probability, which is the product of all these conditional probabilities.

- when t_{prod} is low, rising the number of customers (t_{cust}) increases the accuracy. In fact, as the features (the purchases of the products) are more pertinent when we stick to half of the products' purchases, each customer brings relevency in the data. Each customer's purchase brings significance to determine the real density of probability to purchase a given product.

- when t_{cust} is low, rising the number of products (t_{prod}) blinds the algorithm. This is simply the opposite of the last two points. The products in the tail tend to bring sparsity and bias the conditional probabilities. Therefore, the most biased the probabilities, the less accurate the algorithm.

To sum up the Naive Bayes algorithm gives a very good performance with few relevant products. On the contrary the algorithm explodes when the number of products increases, due to the fact that the number of 0 for the products in the tail overwhelm the number of 1.
To understand better why the algorithm has such different performances according to the tail factors, it is compelling to look at Figures 5.21, 5.22 and 5.23. They all plot the 8 densities of the log-Likelihood function for each of the 8 recommendations within each coupon.
For the chart corresponding to the best NB, with $t_{cust} = 100\%$ and $t_{prod} = 50\%$, the densities are generally close to 0 and stay in a relatively small range. This means that the ratio of the probability to purchase by the probability not to purchase is of reasonable value. However for the second chart, the ratio start to bump and for the last chart it completely explodes. In the last case, the NB recommends only products that have never been bought in all other retailers than Retailer 1. In concrete, these probabilities would be zero if there was no add-one smoothing.
This shows the limit of the Naive Bayes and can help understand why an implementation of NB does not work.

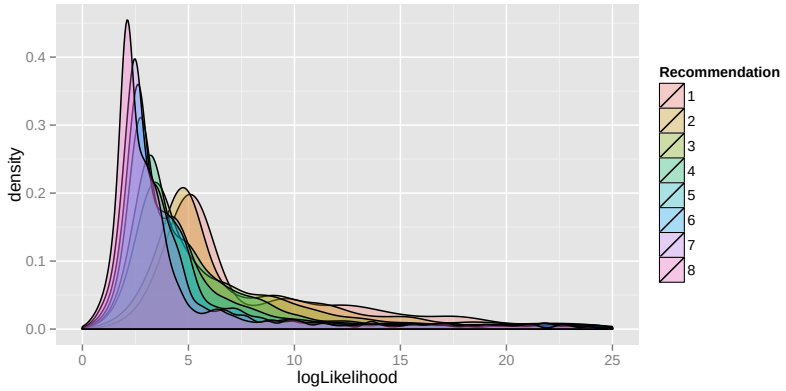


Figure 5.21: Naive Bayes $t_{cust} = 100\%$ $t_{prod} = 50\%$

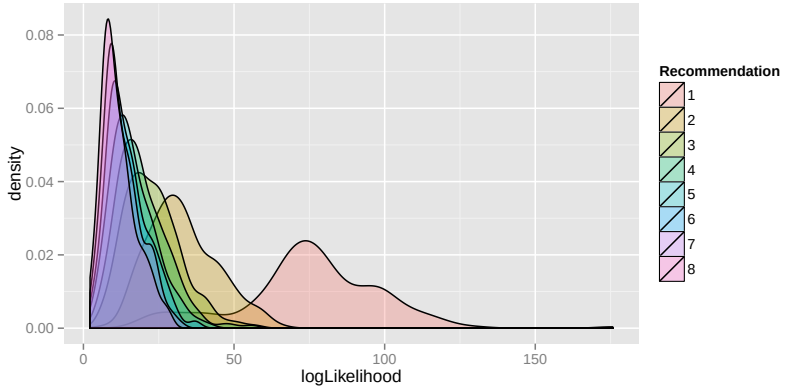


Figure 5.22: Naive Bayes $t_{cust} = 50\%$ $t_{prod} = 50\%$

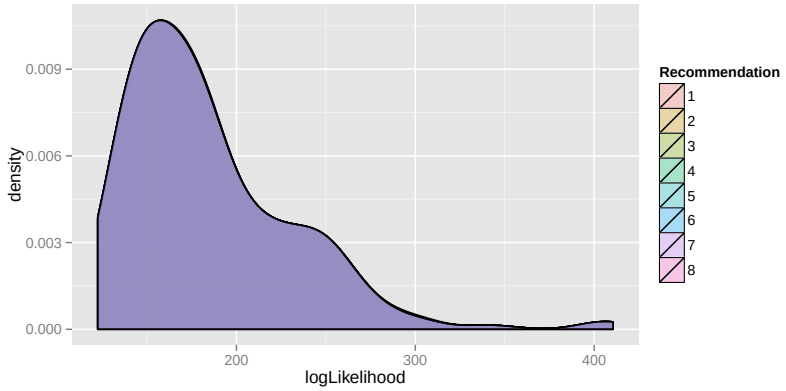
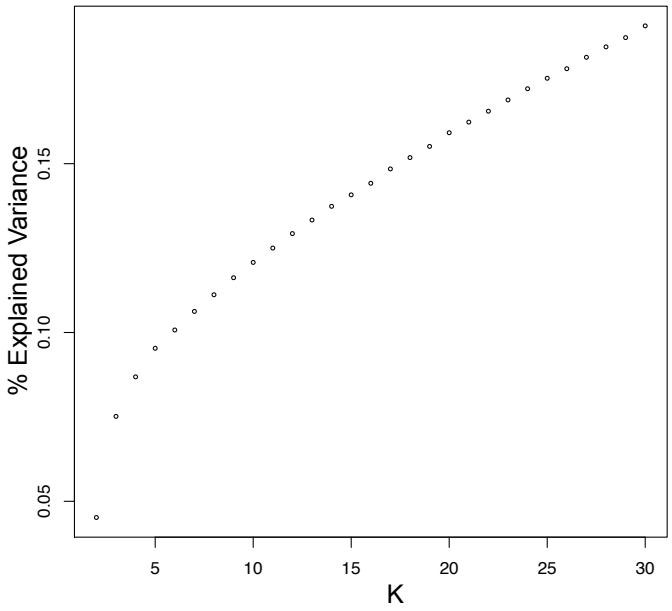


Figure 5.23: Naive Bayes $t_{cust} = 50\%$ $t_{prod} = 80\%$



K-Means

The K-Means algorithm was implemented using the `kmeans` built-in function. The goal was to see whether there existed some obvious clustering which would then be used in combination to the k-NN algorithm. Instead of finding neighbors across all the dataset the idea was to find neighbors within one cluster of customers. This cluster would be found by using the a 1-Nearest Neighbor algorithm across the K points built by the K-Means algorithm.
If the K-Means method always return a partition of clusters, it does not necesarily mean that the clustering is good. Indeed, the initial position of the K centers induce different results most of the time. This is why I used to repeat 20 times (abitrary number that is validated empirically) the random start in the algorithm for a given K. This is easily managed since it this iteration is already built in the function: the parameter `nstart` is set up to 20. On top of that, since the optimal K is unknown, it has to be flexible in the beginning.
The Figure 5.2.3 plots the explained variance of the clusters (the y-axis) as a function of the number of centers K (the x-axis). The chart reveals the real inaccuracy of the K-Means algorithm in this problem. The proportion of the explained variance starts to be straight from $K = 5$ meaning that there is no real improvement from this point. And with $K = 5$ the proportion of

explained variance is around 10%, which indicates that only 10% of the data is explained by this clustering. This is sufficently low to urge to stop digging in that direction.
Actually the issue is no doubt the same encountered when building the first k-NN algorithm: the distance function is of no pertinency here.

5.2.4 Out of sample Results

This sections serves as a final phase in the process of building a Machine Learning algorithm. After building a model trained on a specific sample, it has to be tested on an unknown dataset, which is the test set.
What we observe is actually very similar to what happened on the training set. The 'top8' strategy performs the same (it would have been strange otherwise), the kNN^1 gets a poor performance, the kNN^2 is again very good and the Naive Bayes infers the same consequences regarding the different tail factors.
The Figures 5.24, 5.25, 5.26 and 5.27 display the rate of good predictions for the 4 algorithms. The high performance of both kNN^2 and NB remain.

The goal is then achieved: outperforming the benchmark by a significant improvement for both kNN^2 and NB, while having a high diversity. An interesting chart to plot is the number of times a product is recommended per product for all the algorithms.
It is staggering to see that even if the Machine Learning algorithms give some diversity in the recommendations, they seem to provide a very limited number of different recommendations to the customers. It is also interesting to see that between the two kNN, only the choice of the products makes the difference in accuracy. They seem to recommend products in the same way (red and green lines), yet the products recommended are different. Therefore the distance function seems to play a role only in the choice of the products, but not on the overall process of recommendations. The diversity led by a kNN algorithm seems to be more exogenous than endogenous, which suggests that there is no possible improvements of the diversity when using this algorithm.
The Naive Bayes algorithm provides on the contrary a large diversity of recommendations, or should I say, a long tail in the recommendations. This method gives a fairly similar diversity for the main products than the kNN and 'top8', yet develops a long tail which might infer some serendipity, which is often advocated by Machine Learning practitioners.

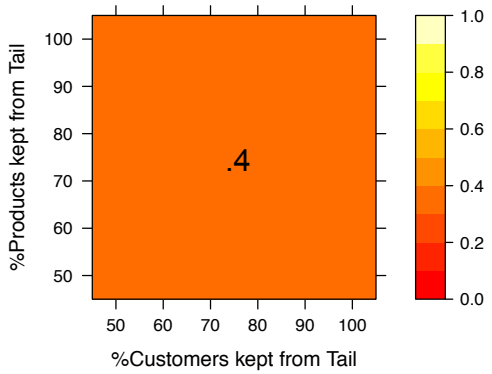


Figure 5.24: Rate of good predictions top8 algorithm

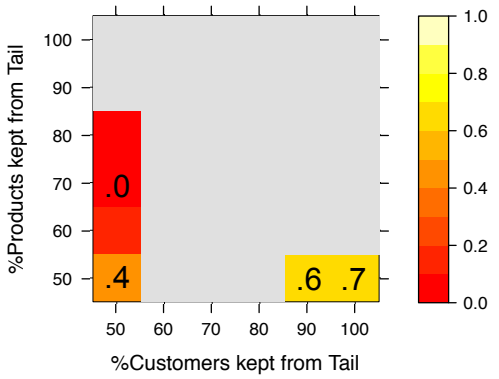


Figure 5.25: Rate of good predictions Naive Bayes algorithm

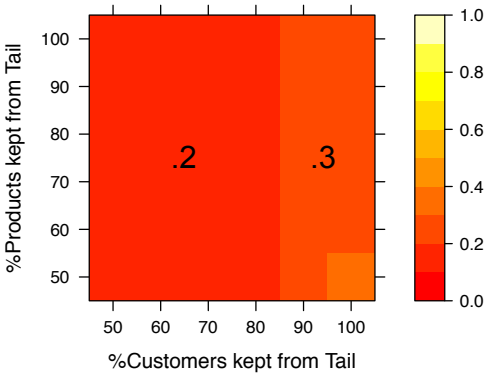


Figure 5.26: Rate of good predictions kNN[1] algorithm

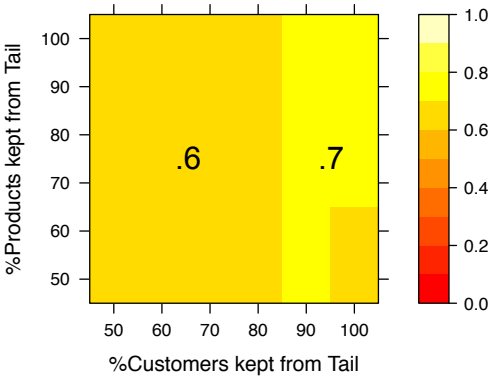


Figure 5.27: Rate of good predictions kNN[2] algorithm

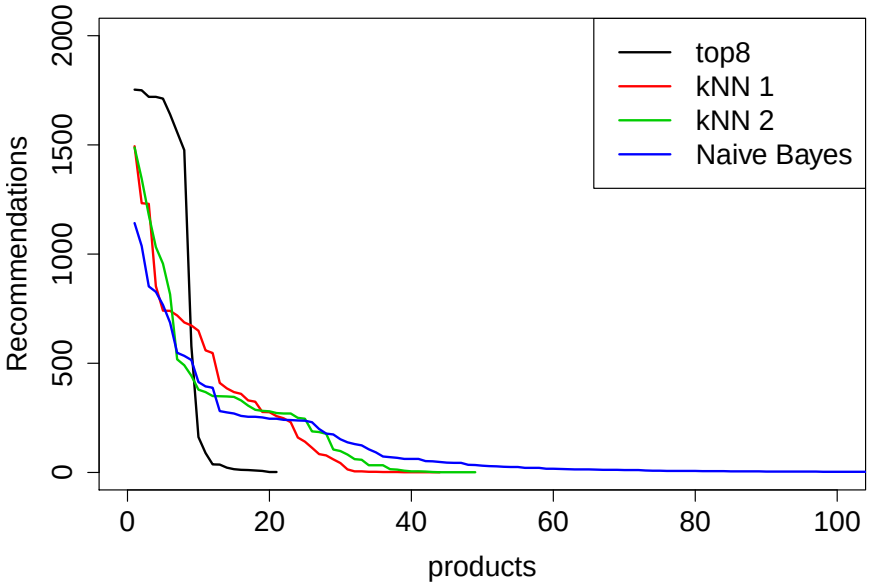


Figure 5.28: Diversity of the recommendations In Sample

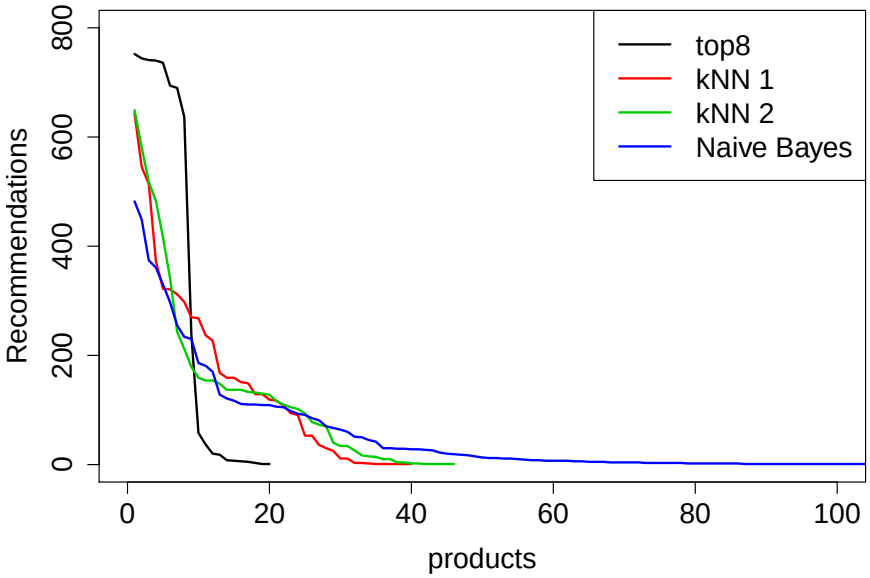


Figure 5.29: Diversity of the recommendations Out of Sample

6 Conclusion & Room for Improvements

Recommending products to a customer is a very challenging purpose for Machine Learning scientists. It requires a deep knwoledge of the available techniques in order to avoid facing a dead end. Moreover, computer skills are highly valuable to gain time for obtaining results. In short, it is very demanding but of is of the utmost interest for statisticians.
In this work, only a few algorithms were actually tested. Still, many different techniques could also be useful for this problem. The section 4 describes a non exhaustive list of Machine Learning tools that can be applied. Some of them are more fitted to the problem than others, but simple manipulations on the data could make the preceeding statement wrong. There are infinite possibilities to attack the problem. This work tried to present an objective analysis of the data and of the available tools, and a modest yet thorough way to solve it.
Some improvements could be applied directly on the algorithms tested in this work. I will give for each of the algorithm some specific lines of refinement, then enlarge the discussion for more open methods.
The kNN algorithm performed very well with the second similarity function, but not with the first one. This suggests that the choice of the distance makes most of the final results. As a consequence, there could be many possibilities for refining the second similarity function. For instance, for the customers who did not buy much in Retailer 1, the neighborhood have a good chance to contain many neighbors. Restricting it to only k neighbors chooses abritrarily between them, and might not be the most efficient choice. A way to solve that could be to select the neighbors who have a number of purchases in Retailer 1 similar to the number of purchases of the customer. Also, when the neighbors have very different behaviors in the other retailers,

the algorithm would not predict very well. I suggest then to use the 'top8' method in these cases, which seems safer for the recommendations.
Regarding the Naive Bayes implementation, different suggestions come in mind. One of the main drawback is the time complexity. On top of the parallel computation, I found out that it would shrewder to compute each conditional probability separately, therefore $2 \times M$ calculations. Then, the time complexity would only lie on the rapidity to retrieve this data, which is much facter than to compute everything again. Another issue is the majority of products that brings sparsity, which completely blinds the algorithm. The way I tackled it in my implementation was to cut the tail. Unfortunately I see no other smart way than this trick.
To evoke other possible refinements, I could cite using a linear blend of the previous algorithms which should improve the accuracy but add complexity. It should be very interesting to use the Decision Tree model in order to understand somehow the structure of the data in addition to predict the purchases. Also, considering time in the data would be of great asset. For instance people purchase regularly in supermarkets. And the algorithm could take that into account: if somebody seems to have bought a product regularly before and now nothing, maybe this customer is now buying it somewhere else. Therefore the recommendation on this product would be a key asset to keep make this customer purchase this product again.
Many discussions arise, but in the end, there is the question of efficiency of the RS. How to know if the coupons worked ? Somebody may have a very accurate coupon, but will this infer purchases ? How to know if all the efforts to build the RS are worthy ? About 70% of good predictions with both $k\text{NN}^2$ and NB seems good enough to proceed. Anoth question is that once a customer have purchased the most recommended products (Figure 5.28), will the next coupons remain accurate ? For the customers who purchase a lot in Retailer 1, is there some value added for them ? This problem seems to address only to customers who purchase irregularly at Retailer 1. Finally, a similar RS developed in a competitor would lead to a cutthroat price competition.
To be complete, the RS should not only focus on the products that have the highest chance to be purchased in another retailer, but should also suggest to a customer a variety of products that similar customers have purchased in Retailer 1 for example.

Bibliography

[1] Xavier Amatriain. Recommender systems. In *MLSS'14 Pittsburgh*, 2014.

[2] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2013.

[3] J. Beel, S. Langer, M. Genzmehr, B. Gipp, and A Nürnberger. A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation. In ACM, editor, *RepSys '13 Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, pages 7–14, 2013.

[4] David J. Hand. Classifier technology and the illusion of progress. *Statistical Science*, 21(1):1–15, June 2006.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[6] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender System, An Introduction*. Cambridge University Press, 2011.

[7] Kevin P. Murphy. *Machine Learning, A Probabilistic Perspective*. The MIT Press, 2012.

[8] Ricci, F. Rokach, L. Shapira, B. Kantor, and P.B. *Recommender Systems Handbook*. Springer, 2011.

[9] Martin P. Robillard, Walid Maalej, Robert J. Walker, and Thomas Zimmermann. *Recommendation Systems in Software Engineering*. Springer, 2014.